2016 IEEE 40th Annual Computer Software and Applications Conference

An Assessment of Internet of Things Protocols for Resource-Constrained Applications

Dae-Hyeok Mun, Minh Le Dinh, and Young-Woo Kwon
Department of Computer Science
Utah State University, USA
Email: {daehyeok, minh.le}@aggiemail.usu.edu and young.kwon@usu.edu

***Abstract*—With the resource constraints of Internet of Things (IoT) devices, performance and energy efficiency have become an important IoT software design consideration while minimizing resource usages. To provide efficient communication for resource-constrained applications, several IoT protocols have been introduced, and yet there are few existing guidelines to help the programmer choose an appropriate communication protocol. To address this limitation, we evaluate five different communication protocols including CoAP, MQTT, MQTT-SN, WebSocket, and TCP. Our result can help IoT application programmers make informed decision when choosing communication protocols for their applications.**

***Index Terms*—internet of things; protocols; assessment; performance; energy efficiency; resources**

I. Introduction

The next couple of years will see the vision of the Internet of Things (IoT)—environmental or daily items, including *physical things*, *objects*, and *machines*, enhanced to provide intelligence and connectivity through the Internet—entering the computing mainstream. According to Gartner and ABI Research, 26 billion or more than 30 billion devices will be connected to the Internet by 2020 [1], [2]. By communicating, interacting, and sharing information, *physical things* can have the intelligence to infer *Context—"any information that can be used to characterize the situation of entity including a person, place, or object that is considered relevant to the interaction between a user and an application"*[3].

However, when trying to realize this vision, we often face new types of technical challenges including building a software stack for heterogeneous *physical things*, organizing wireless networks, designing energy- and performance-efficient applications. Specifically, IoT applications are usually operated in wireless networks, in which they are executed in a variety of environments that differ in terms of their respective network conditions and limited hardware resources. For example, the execution demands of small wireless IoT devices are outstripping their battery lives. Thus, energy efficiency (i.e., fitting an energy budget and maximizing the utility of applications under given battery constraints), performance-efficiency, availability (e.g., utilization) have become an important software design consideration. As a result, for the efficient (i.e., energy- and performance-efficiency) executions of IoT applications and services, IoT application programmers must choose an appropriate network protocol for their IoT devices and applications.

Several IoT network protocols have been introduced, and yet there are few existing guidelines to help the programmer choose an appropriate IoT network protocol to achieve desired goals for performance, reliability, energy efficiency, and expressiveness. To address this limitation, we evaluate three IoT protocols including Constrained Application Protocol (CoAP) [4], Message Queuing Telemetry Transport (MQTT) [5], MQTT-Sensor Network (SN) [6], and two alternatives including TCP and WebSocket [7]. We compare and contrast these protocols, which have been implemented as open-source projects. By measuring performance, energy efficiency, and resource usages (CPU and memory), we want to be able:

1) to understand different IoT protocols' execution patterns under different network conditions and message sizes.
2) to help IoT application programmers make informed decision when choosing network protocols for their resource-constrained applications.
3) to infer opportunities for improving the energy efficiency and performance of emerging IoT protocols.

The rest of this paper is structured as follows. Section II introduces the technologies used in this work. Section III describes our experimental results. Section IV discusses related work, and Section V presents future

0730-3157/16 $31.00 © 2016 IEEE
DOI 10.1109/COMPSAC.2016.51

research directions and concluding remarks.

II. Technical Background

In the following discussion, we describe three IoT protocols protocols we have evaluated in this study.

A. Constrained Application Protocol

The Constrained Application Protocol (CoAP) [4] has been designed by Internet Engineering Task Force (IETF) to facilitate message transfer between machine-to-machine (M2M) applications by offering various features such as built-in discovery, multicast support, and asynchronous message exchanges. The main goal of CoAP is to design a web protocol on top of UDP for special environments that consist of constrained nodes (e.g., resources, computing power) and networks (e.g., low-power, lossy). There are more than 30 CoAP implementations[1] written in various languages including C, C++, Java, Python, JavaScript, etc.

B. Message Queuing Telemetry Transport

The Message Queuing Telemetry Transport (MQTT) [5] is a standard protocol of Organization for the Advancement of Structured Information Standards (OASIS) standard. lightweight publish/subscribe messaging transport application level protocol. It uses TCP/IP protocol and designed for machine-to-machine and IoT purpose. MQTT-SN be designed to use similar functionality in the wireless communication environment considering low bandwidth. Unlikely MQTT, MQTT-SN use UDP. Recently, Amazon started to support MQTT in their Amazon Web Services[2].

C. WebSocket

The WebSocket Protocol [7] also has been designed by IETF to provide full-duplex communication between the web browser and server to overcome the existing bidirectional communication technologies using HTTP while supporting existing HTTP infrastructures over HTTP ports—80 and 443. Although the WebSocket Protocol has been designed to supersede existing HTTP technologies, it recently has been used for the IoT applications that require real-time communication.

[1]http://coap.technology/impls.html

[2]https://aws.amazon.com/iot/

III. Experiments

For the purpose of this study, we define our metrics as follows:

- Performance: the total transmission time it takes to send messages and receive their acknowledgments.
- Energy efficiency: the total consumed energy for the given execution time.
- Resource usage: the average Java heap usage measured per 500 ms and the total CPU time including user- and kernel-mode time.

In this benchmark, we compare these metrics for five protocols: (1) CoAP, (2) MQTT, (3) MQTT-SN, (4) WebSocket, and (5) TCP.

A. Experimental Setup

Our experimental setup comprised a client and three servers. For the servers, we have set up one local and two Amazon Web Services EC2 micro instances in Oregon and Tokyo. All the servers run Ubuntu 14.04.2 LTS, JVM build 1.8.0. on the both local server[3] and two remote servers[4]. The client is a Raspberry Pi 2 model B[5] running Debian GNU/Linux 7.8 (wheezy) and JVM build 1.8.0, connected via a wireless LAN. To measure energy consumption, we have used Monsoon Power Monitor, a hardware based power profiling tool.

TABLE I
Library and framework used for experiment

Protocol		Library
TCP	Client	java.nio.channels API
	Server	Twisted framework(13.2.0)
WebSocket	Client	javax.websocket API
	Server	Python Autobahn framework (0.10.9)
CoAP	Client	Eclipse Californium Framework (1.0.0-M3)
	Server	Eclipse Californium Tools (1.0.0-M3)
MQTT	Client	Eclipse Paho Client (3_1.0.2)
	Server	Mosquitto (1.4.5)
MQTT-SN	Client	Eclipse Paho mqttsn (GIT commit id 0768b17)
	Server	Eclipse Paho mqttsn (GIT commit id 0768b17)

Then, we selected widely used third-party IoT protocol implementations. Table I summarizes the libraries and their versions we used in the benchmarks. In particular, we have used (1) java.net package for TCP, (2) javax.websocket package for Websocket, (3) Californium[6] for CoAP, and Paho[7] for MQTT/MQTT-SN, respectively. All the libraries are provided by the IoT working group of Eclipse Foundation. We have experimented

[3]Intel(R) Core(TM) i7-3770 CPU @ 3.40 GHz (quad-core), 16 GB RAM

[4]Intel(R) Xeon(R) CPU E5-2670 v2 @ 2.50GHz 1vCPU, 1 GB RAM

[5]1GB & 8GB NOOBS Edition

[6]http://www.eclipse.org/californium/

[7]http://www.eclipse.org/paho/

without any modification on library except MQTT-SN. However, because none of third party MQTT-SN implementations does not support transferring large messages, we modified the Paho library (i.e., version 1.3) to publish a large message up to 65,535 bytes.

Algorithm 1 shows how we experimented with each protocol to assess performance, resource usages and energy efficiency on the client side. We do not measure the energy consumed by the server. First, to warm the JVM, the benchmark application sends small dummy data three times and then sends a randomly generated message to a remote server using different packet lengths ranging from 128 bytes to 1920 bytes. To measure the CPU and Java heap usage, we used jvmTop [8] and the `\proc\[pid]\stat` file, respectively.

Algorithm 1 Benchmark procedure for each protocol

```
for all target_size in [128, 256, 512, ... 1920] do
    Connect to Server
    for i = 0 to 3 do
        send 5Bytes data
    end for
    start power monitoring
    start_cpu ← copy proc/pid/stat
    message ← 50Kb random string
    for i = 0 to 20 do
        start_time ← current time
        repeat
            packet ← read target_size from message
            send packet
        until message is empty
        end_time ← current time
        trans_time ← start_time − end_time
    end for
    Disconnect to Server
    end_cpu ← copy proc/pid/stat
    stop power monitoring
end for
```

We have experimented with three local and remote servers that have the following respective network characteristics. As described in Table II,

TABLE II
NETWORK STATUS ON EACH SERVER

	Local	Oregon	Tokyo
TCP Bandwidth (Mbits/s)	27.1	26.9	2.14
UDP Bandwidth (Mbits/s)	52.7	51.4	50.80
Jitter (ms)	0.884	0.670	0.598
RTT (ms)	6.635	30.354	144.837

B. Experimental Results

1) Performance: Figure 1 compares the performance of five protocols with respect to the average transmission time when sending a 50kB message to three servers using different packet lengths. As expected, the plain TCP implementation consistently showed the best performance across all benchmarks, but surprisingly the WebSocket Protocol also showed the similar results. MQTT-SN showed the poorest results in the local and Oregon cases while the performance of MQTT improves as the packet size increases. Because MQTT-SN uses UDP as a transport protocol while MQTT uses TCP, MQTT-SN is particularly vulnerable to limited network conditions than MQTT. Even though CoAP is a UDP-based protocol, it outperforms MQTT and MQTT-SN when its packet size is less than 1024 bytes. When the packet size of CoAP becomes larger than 1024 bytes, CoAP internally fragments the large packet into small blocks, thereby raising up the total transmission time. Moreover, due to the lack of retransmission mechanisms in UDP, MQTT-SN and CoAP easily become inefficient under the limited network conditions as depicted in Figure 1-c. Thus, when the packet size is relatively large, MQTT should be favored over CoAP.

2) Energy Efficiency: In this benchmark, we assess the energy efficiency of each protocol. To that end, we first measured the total amount of energy consumed by each protocol during the entire benchmark. Limited network conditions such as latency, jitter, packet loss rate are usually known to increase the energy cost of network processing [9]. However, TCP and WebSocket protocols are not affected by servers' physical locations that have different delay/bandwidth characteristics as described in Table II. In contrast, the energy efficiency of CoAP, MQTT, and MQTT-SN is considerably vulnerable to network conditions due to the underlying transport protocols (TCP or UDP), frequent packet fragmentation (CoAP), straightforward retransmission mechanisms (CoAP, MQTT, and MQTT-SN), poor implementation (MQTT-SN), etc.

TABLE III
TOTAL ENERGY CONSUMPTION OF EACH PROTOCOL (MJ)

	Local	Oregon	Tokyo
TCP	457.70	468.08	478.60
Websocket	574.98	588.97	611.87
CoAP	953.42	2414.27	8484.18
MQTT	1705.20	2841.90	8256.88
MQTT-SN	3266.48	4187.13	9393.42

Then, we broke down the total energy consumption into the amount of energy consumed when using different packet sizes. Figure 2 shows the amount of energy consumed by each protocol when varying the packet size.

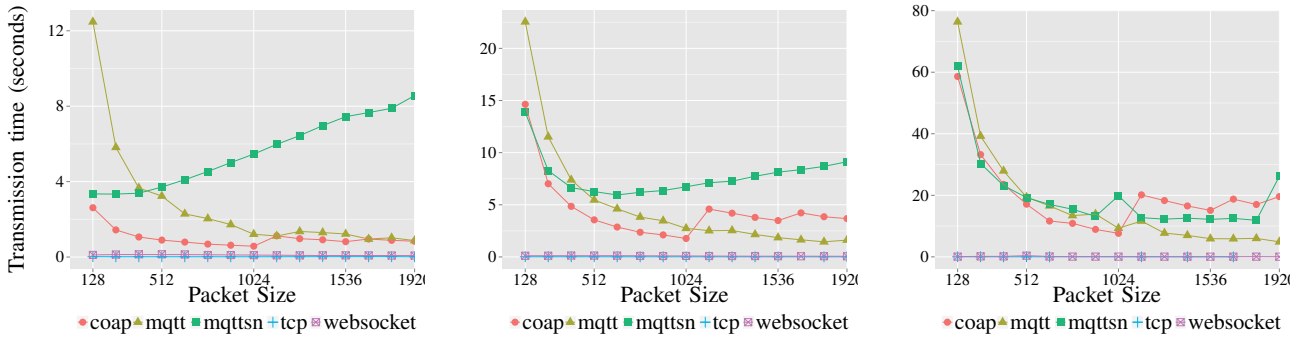

Fig. 1. Performance comparison (a. Local, b. Oregon, c. Tokyo)

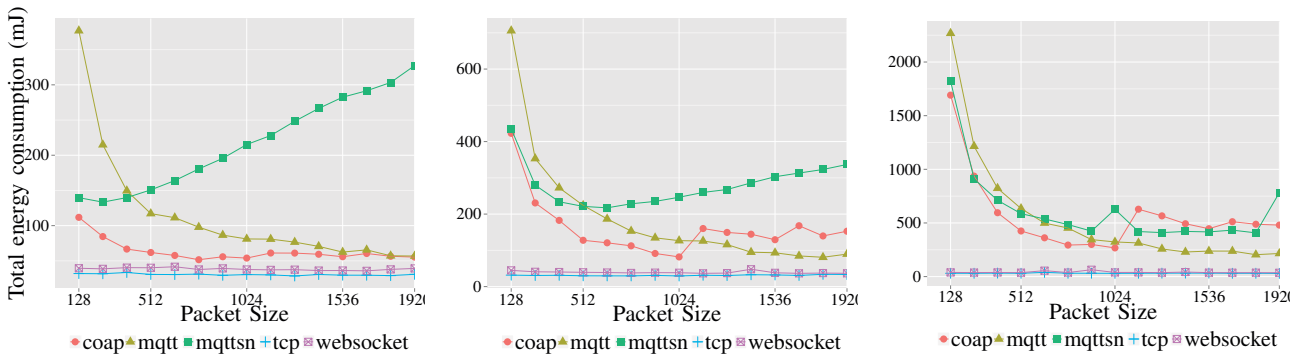

Fig. 2. Energy consumption comparison (a. Local, b. Oregon, and c. Tokyo)

Similar to the performance characteristics, MQTT-SN showed the poorest results while TCP and WebSocket showed the best results. Similarly, CoAP's energy consumption increases when the packet size becomes larger than 1024 bytes due to the fragmentation. When the packet size of MQTT is larger than 1024 bytes in Oregon and Tokyo cases, it consumes less energy than CoAP. Thus, when the packet size is relatively large, MQTT should be favored over CoAP.

3) Resource Usage: CPU and Java Heap: Next, we assess resource usages in terms of CPU and memory. As IoT applications run on resource-constraint small devices, the resource usage of network protocols should be considered. Figure 3 shows the CPU time and and Java heap usage during the test. The CPU time is computed as follows: $utime + ptime$ of `\proc\[pid]\stat`. Similar to the performance and energy efficiency characteristics, the CPU usage of MQTT-SN significantly increases especially when the client is connected to the local and Oregon server and the packet size increases.

To assess memory usage, we have measured the Java heap size of each protocol. As depicted in Figure 3-a, b, and c, while the Java heap usage of TCP and WebSocket is consistent regardless the physical location of servers, CoAP, MQTT, and MQTT-SN consume more memory spaces. In particular, CoAP requires more memory space than other protocols when the client is connected to Oregon and Tokyo servers. However, we could not observe the similar patterns shown in performance, energy consumption, and CPU time experiments due to fragmentation. This experimental results show that CoAP's retransmission mechanism mainly affected the memory usage rather than the CPU usage.

C. Results Analysis

In this section, we discuss some of the implications of the performance, energy efficiency and resource usage measurements presented above.

First, regardless of the distance between a client and a server, our experiments show that MQTT and CoAP protocols are largely affected by the packet size. In particular, CoAP and MQTT start to fragment a packet from 1 kB and 1.5 kB respectively. In other words, if an application utilizes the MQTT protocol, the packet size should be up to 1.5 kB for the performance and energy efficiency. If a programmer wants to use REST-like APIs,

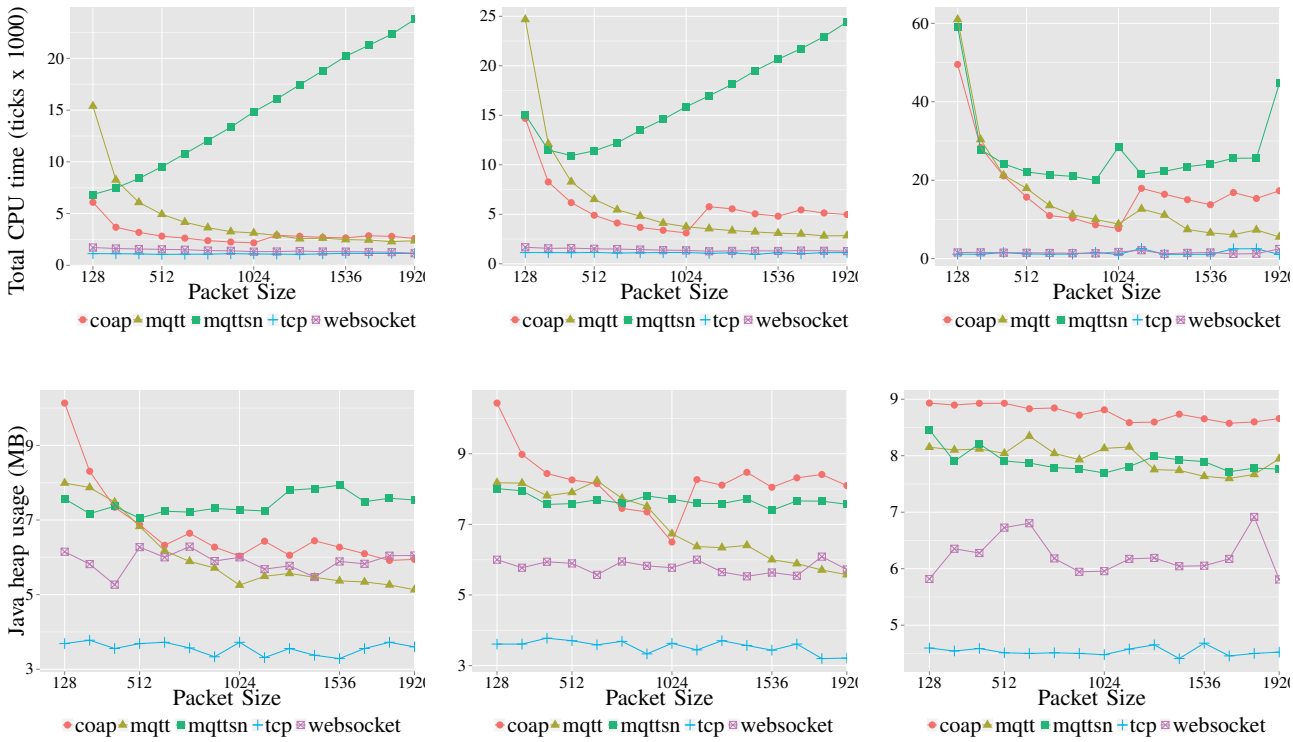

Fig. 3. CPU and Java heap usage comparison (a. Local, b. Oregon, c. Tokyo, d. Local, e. Oregon, and f. Tokyo).

the packet size of CoAP should be less than 1 kB to avoid performance/energy overheads caused by the error recovery. In particular, in order to recover errors due to packet losses or delays, CoAP employs a straightforward retransmission mechanism using exponential backoff time on top of UDP while MQTT relies on the TCP's error recovery mechanism. Furthermore, due to poor implementation, MQTT-SN is not ready for real-world mobile software development. Finally, because TCP and WebSockets often outperform IoT protocols,

D. Threats to Validity

The experimental results are subject to both internal and external validity threats. The internal validity is threatened by the way in which we chose to use one library per one protocol to compare protocols. Experiment results, especially resource and energy consumption affected by their implementation. Therefore it has possibility that other library and framework implement the protocol with more efficient or inefficient way.

The way we used these protocols in the benchmarks may not be fully optimal, in terms of the configuration options specified. We use default configuration proved by official distribution or documents. It is likely that a programmer deeply experienced in any of the measured library may implement the same functionality exhibiting the performance and resource/energy consumption patterns differing from our observations.

The external validity is threatened by our measurement infrastructure. Rather than use the restricted environment by manual configuration, we choose cloud server physically locates a different region. However, in the most our test cases, Oregon and Tokyo have a similar pattern. So we can expect other remote server will have similar results with our experiment.

IV. Related Work

To the best of our knowledge, this work is the first attempt to assess the performance, energy consumption, and resource usage characteristics of IoT protocols. However, several prior efforts have informed and inspired this work.

Two IoT protocols—CoAP and MQTT—have been assessed in terms of energy consumption, bandwidth utilization and reliability [10]. According to the result, CoAP is the most efficient in terms of energy consumption and bandwidth usage while MQTT provides high reliability. Niccolo De ro et al. [11] also compare these

two IoT protocols with multiple application scenarios and then suggest that MQTT is more appropriate for applications that require advanced functions such as message persistence, secure multicast and reliability. In addition, Matteo Collina et al. [12] measure performance of these protocols with different packet lost rates and delay/bandwidth characteristics. According to the study, MQTT offers better throughput than CoAP in congested networks. Moreover, Lars Durkop et al. [13] characterized the transmission behaviors of IoT protocols—MQTT, CoAP and OPC UA (Open Platform Communication United Architecture)—in the emulated networks including GSM (2G), UMTS (3G) and LTE (4G). Finally, MQTT and CoAP have been assessed using a middleware system that can accommodate different protocols [14]. In these studies, IoT protocols have been assessed in terms of performance, energy consumption or both while our study systematically compare five different protocols in terms of performance, energy consumption, and resource (i.e., CPU and memory) usages.

Finally, in our prior work, we assessed different middleware mechanisms in terms of performance, energy consumption, reliability, and programming effort [9], [15]. In particular, after discovering that the network conditions can significantly affect how much energy is spend in different middleware mechanisms, we created guildelines for mobile application programmers to choose an appropriate middleware mechanism in a fashion that consumes the least amount of energy or takes the shortest time for a given mobile network.

V. Conclusion and Future Work

The IoT application programmers are faced with the challenges of choosing an appropriate communication protocol for their resource-constrained applications. To assist the programmers in their decision process, in this paper, we compared the performance, energy efficiency, and resource usages of five different network protocols.

As a future work, we plan to conduct more systematic measurements of other IoT protocols with a wider range of user scenarios including reliability, security, expressiveness (e.g., lines of code, complexity), etc. We also plan to create systematic guidelines to help IoT application programmers make informed decisions when choosing IoT protocols for their resource-constrained applications.

Acknowledgments

The authors would like to thank Dr. Ji-In Kim for his help in designing and setting up the experiments. This research is supported by the Utah State University Research Catalyst Grant.

References

[1] Gartner, Inc., "Gartner Says the Internet of Things Installed Base Will Grow to 26 Billion Units By 2020," Dec. 2013.

[2] ABB Research, "30 Billion Devices Will Wirelessly Connect to the Internet of Everything in 2020," May 2013.

[3] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a better understanding of context and context-awareness," in *Handheld and ubiquitous computing*, 1999.

[4] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (CoAP)," 2014.

[5] OASIS, *MQTT Version 3.1.1*, Std., 2014.

[6] A. Stanford-Clark and H. L. Truong, "Mqtt for sensor networks (MQTT-SN) protocol specification version 1.2," 2013.

[7] I. Fette and A. Melnikov, "The Websocket Protocol," 2011.

[8] patric-r/jvmtop. [Online]. Available: https://github.com/patric-r/jvmtop

[9] Y.-W. Kwon and E. Tilevich, "The impact of distributed programming abstractions on application energy consumption," *Information and Software Technology*, 2013.

[10] A. Bhattacharyya and S. Bandyopadhyay, "Lightweight internet protocols for web enablement of sensors using constrained gateway devices," in *Proceedings of the 2013 International Conference on Computing, Networking and Communications (ICNC)*, 2013.

[11] N. De Caro, W. Colitti, K. Steenhaut, G. Mangino, and G. Reali, "Comparison of two lightweight protocols for smartphone-based sensing," in *2013 IEEE 20th Symposium on Communications and Vehicular Technology in the Benelux (SCVT)*, 2013.

[12] M. Collina, M. Bartolucci, A. Vanelli-Coralli, and G. Corazza, "Internet of things application layer protocol analysis over error and delay prone links," in *2014 7th Advanced Satellite Multimedia Systems Conference and the 13th Signal Processing for Space Communications Workshop (ASMS/SPSC)*, 2014.

[13] L. Durkop, B. Czybik, and J. Jasperneite, "Performance evaluation of m2m protocols over cellular networks in a lab environment," in *2015 18th International Conference on Intelligence in Next Generation Networks (ICIN)*, 2015.

[14] D. Thangavel, X. Ma, A. Valera, H.-X. Tan, and C.-Y. Tan, "Performance evaluation of mqtt and coap via a common middleware," in *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP),*, 2014.

[15] Y.-W. Kwon, E. Tilevich, and W. R. Cook, "Which middleware platform should you choose for your next remote service?" *Service Oriented Computing and Applications*, vol. 5, no. 2, pp. 61–70, 2011.